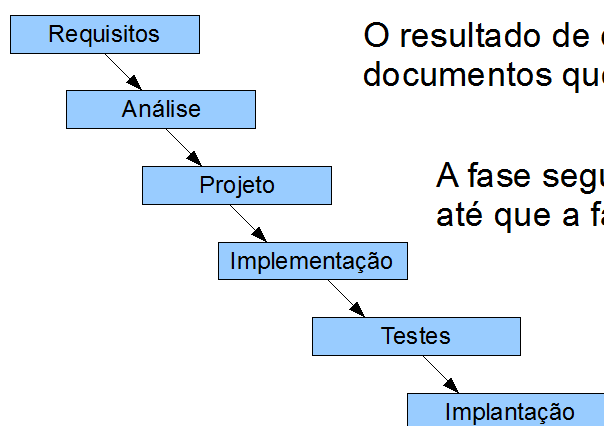
## Atividades típicas do processo de desenvolvimento

- Elicitação de Requisitos
  - Qual o problema?
  - O que será feito?
- Análise e projeto de software
  - Como será feito?
  - Descrição computacional
  - Projeto de arquitetura e detalhado
- Implementação
- Testes
- Implantação



## Modelo Linear (Cascata/Waterfall)

Requisitos → Análise → Projeto → Implementação → Testes → Implantação

O resultado de cada fase envolve documentos que devem ser validad

A fase seguinte não deve inicia até que a fase anterior termine

Problema: inflexível divisão em fases distintas

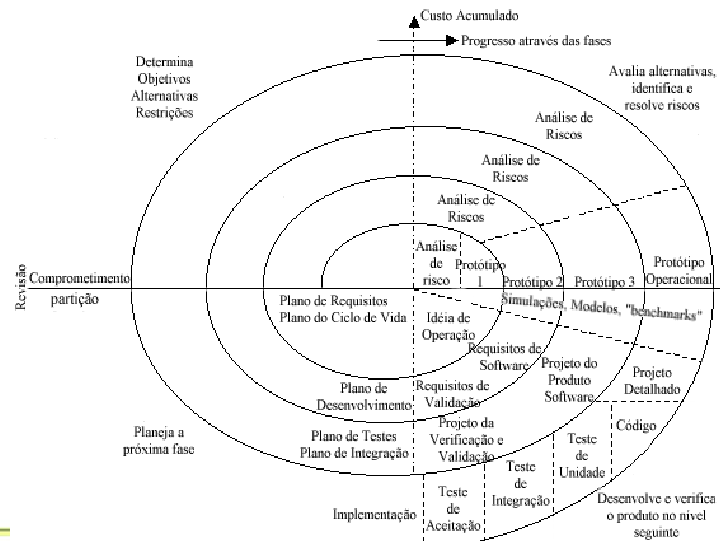
Funciona apenas quando os requisitos forem bem entendidos

## Modelo Linear - crítica

- Projetos reais raramente seguem o fluxo sequencial
- Em projetos grandes, é impossível elicitar todos os requisitos antes da fase de análise
- Requisitos são frequentemente alterados, o que impacta na análise e projeto
- Demora na descoberta de erros
- Espera por fim de uma fase para iniciar outra (bloqueio)



## Desenvolvimento iterativo e incremental

- Diversos passos (iterações) com adição de funcionalidades (incremental)
- Requisitos são agrupados por funcionalidade e implementados em um ciclo de desenvolvimento
- Partes do software são entregues baseado em requisitos prioritários.
- Requisitos com maiores riscos são atacados primeiro

# Modelo em espiral

- Cada loop é dividido em 4 setores:
  - Definição de objetivos
  - Avaliação e redução de riscos
  - Desenvolvimento e validação
  - Planejamento

# Metodologias ágeis

- conceitos chave do "Manifesto Ágil":
  - **Indivíduos e interações ao invés de processos e ferramentas.**
  - **Software executável ao invés de documentação.**
  - **Colaboração do cliente ao invés de negociação de contratos.**
  - **Respostas rápidas a mudanças ao invés de seguir planos.**
- Ex. Extreme Programming (XP), Scrum



# Extreme Programming

- A Extreme Programming (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente.
- Diferenciais
  - *Feedback constante*
  - Abordagem incremental
  - A comunicação entre as pessoas é encorajada
- 12 práticas

## Algumas práticas da XP

- Entregas freqüentes
- Projeto simples
- Programação em pares
- Propriedade coletiva
- Integração contínua
- Código padronizado



## Ferramentas CASE

- Computer Aided Software Engineering
- Ferramentas que auxiliam (dão suporte) as atividades de um processo de desenvolvimento de software
- Ex. editores de modelos, compiladores, depuradores de código, gerenciadores de configuração, ferramentas de teste, ambientes de desenvolvimento

# Uso de ferramentas CASE na disciplina

- StarUML
- ArgoUML
- RationalRose
- Astah
- Qualquer outra que o grupo quiser



## O processo de software

- Um conjunto estruturado de atividades necessárias para desenvolver um sistema de software.
- Existem vários processos de desenvolvimento de software diferentes mas todos envolvem:
  - ✓ especificação – definição do quê o sistema deve fazer;
  - ✓ projeto e implementação – definição da organização do sistema e implementação do sistema;
  - ✓ validação – checagem de que o sistema faz o que o cliente deseja;
  - ✓ evolução – evolução em resposta a mudanças nas necessidades do cliente.
- Um modelo de processo de desenvolvimento de software é uma representação abstrata de um processo. Ele apresenta uma descrição do processo de uma perspectiva em particular.

## Descrições de processo de software

- Quando descrevemos e discutimos processos, geralmente falamos sobre as atividades desses processos, tais como especificação de modelo de dados, desenvolvimento de interface de usuário, etc. e organização dessas atividades.
- Descrições de processos também podem incluir:
  - ✓ Produtos, que são os resultados de uma atividade do processo;
  - ✓ Papéis, que refletem as responsabilidades das pessoas envolvidas no processo;
  - ✓ Pré e pós-condições, que são declarações que são verdadeiras antes e depois de uma atividade do processo ser executada, ou um produto produzido.

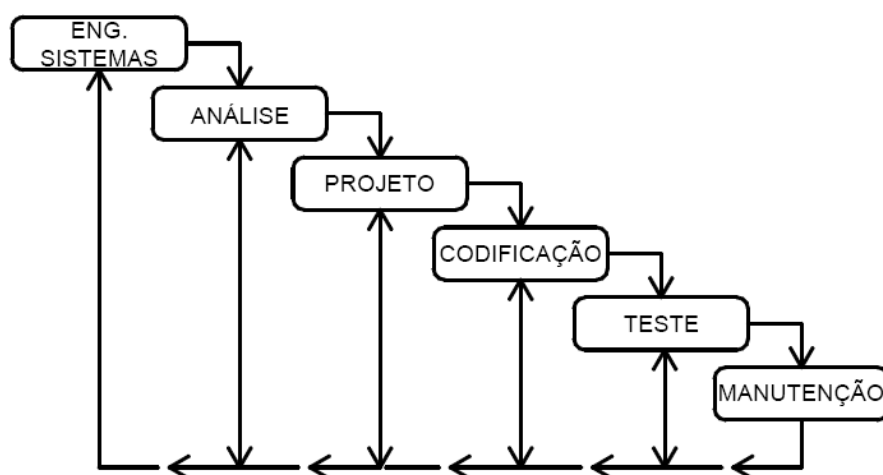


## Processos dirigidos a planos e ágeis

- Processos dirigidos a planos são processos em que todas as atividades do processo são planejadas com antecedência e o progresso é medido em relação a esse plano.
- Nos processos ágeis o planejamento é incremental e é mais fácil modificar o processo para refletir alterações nos requisitos do cliente.
- Na realidade, os processos mais práticos incluem elementos dos processos ágeis e dirigidos a planos.
- Não existe processo de software certo ou errado.

## Modelos de processo de software

- Modelo Cascata – Modelo dirigido a planos. Fases de especificação e desenvolvimento separadas e distintas.
- Desenvolvimento Incremental – Especificação, desenvolvimento e validação são intercaladas. Pode ser dirigido a planos ou ágil.
- Engenharia de software orientada a reúso – O sistema é montado a partir de componentes já existentes. Pode ser dirigido a planos ou ágil.

Na realidade a maioria dos grandes sistemas são desenvolvidos usando um processo que incorpora elementos de todos esses modelos.

# Engenharia de Software

- ENGENHARIA DE SISTEMAS
  - Levantamento dos requisitos
  - Inserir o sistema em um contexto maior –Hardware; Pessoas; Outros sistemas
  - Visão geral e ampla do sistema
  - Riscos; Custos; Prazos; Planejamento



# Engenharia de Software

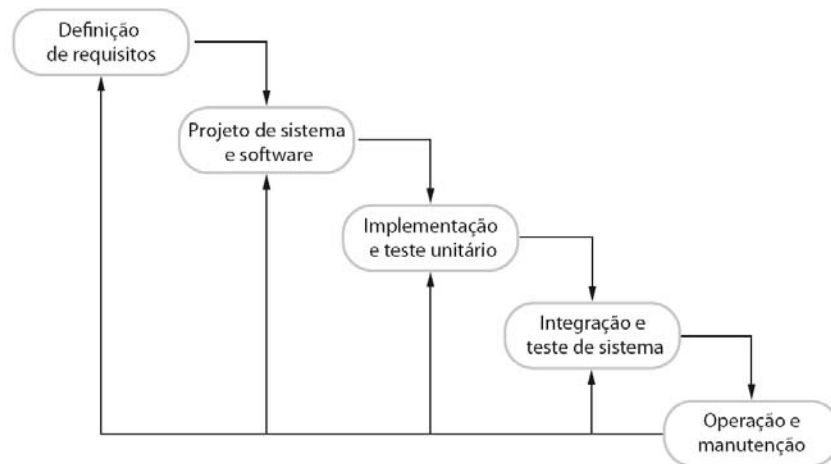
- ANÁLISE
  - Continua o processo de coleta de requisitos, porém concentra-se no âmbito do software
  - Modelos – Dados; Funções e comportamentos
  - Particionamento do problema
  - Documentação e Revisão dos requisitos
    - ANÁLISE ESTRUTURADA – DFD
    - ANÁLISE ORIENTADA A OBJETOS – Diagramas de Caso Uso

## Engenharia de Software

- **PROJETO**
  - “Como” o software irá executar os requisitos
  - Estrutura de dados; Arquitetura do Software;
  - Detalhes de execução; caracterização da interface
  - Produzir um modelo que permita a sua construção posterior
    - PROJETO ESTRUTURADO – Módulos
    - PROJETO ORIENTADO A OBJETOS – Atributos;Especificação dos Métodos; Mensagens
    - Diagramas de Sequência; Diagrama de Classes



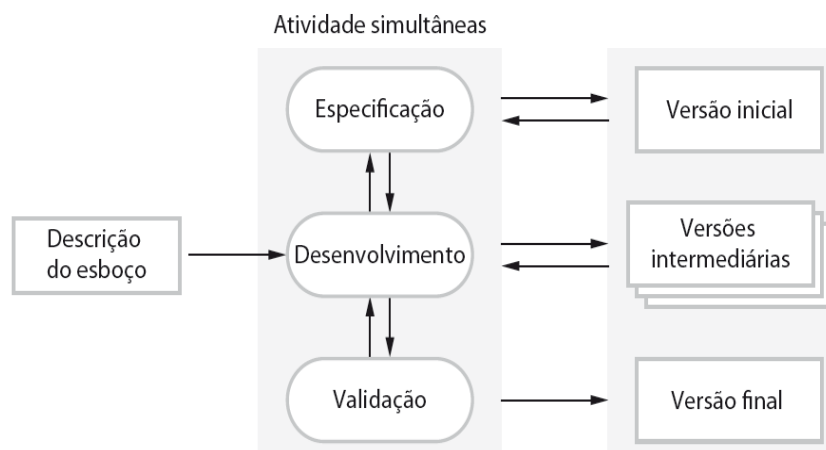
## Engenharia de Software

- **CODIFICAÇÃO**
  - “Traduzir” o projeto para uma linguagem de computador
  - Projeto detalhado pode levar a uma codificação mecânica (Ferramenta CASE)
- **TESTES**
  - Verificação se o código atende aos requisitos
  - Aspectos lógicos e internos do software – Teste de todas as instruções
  - Aspectos funcionais externos – entrada produz o resultado esperado

## Fases do modelo cascata

- Existem fases identificadas e separadas no modelo cascata:
  - ✓ Análise e definição de requisitos
  - ✓ Projeto de sistema e software
  - ✓ Implementação e teste de unidade
  - ✓ Integração e teste de sistema
  - ✓ Operação e manutenção
- O principal inconveniente do modelo cascata é a dificuldade de acomodação de mudanças depois que o processo já foi iniciado. Em princípio, uma fase precisa ser completada antes de se mover para a próxima fase.

Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

## Problemas do modelo cascata

- Divisão inflexível do projeto em estágios distintos torna difícil responder às mudanças nos requisitos do cliente.
  - ✓ Por isso esse modelo só é apropriado quando os requisitos são bem entendidos e as mudanças durante o processo de projeto serão limitadas.
  - ✓ Poucos sistemas de negócio possuem requisitos estáveis.
- O modelo cascata é mais usado em projetos de engenharia de grandes sistemas onde o sistema é desenvolvido em vários locais.
  - ✓ Nessas circunstâncias, a natureza do modelo cascata dirigida a planos ajuda a coordenar o trabalho.



## Desenvolvimento incremental

## Benefícios do desenvolvimento incremental

- O custo para acomodar mudanças nos requisitos do cliente é reduzido.
  - ✓ A quantidade de análise e documentação que precisa ser feita é bem menor do que o necessária no modelo cascata.
- É mais fácil obter feedback do cliente sobre o trabalho de desenvolvimento que tem sido feito.
  - ✓ Os clientes podem comentar demonstrações do software e ver quanto foi implementado.
- Possibilidade de mais rapidez na entrega e implantação de software útil para o cliente.
  - ✓ Os clientes podem usar e obter ganhos do software mais cedo do que é possível no processo cascata.



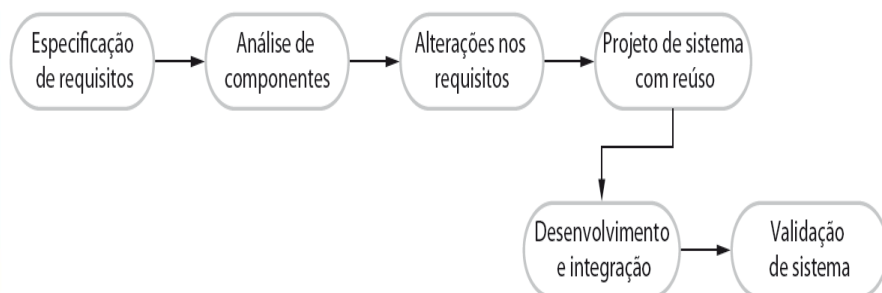
## Problemas do desenvolvimento incremental

- O processo não é visível.
  - ✓ Gerentes precisam de entregas regulares para medir o progresso. Se os sistemas são desenvolvidos de forma rápida, não é viável do ponto de vista do custo produzir documentação para refletir todas as versões do sistema.
- A estrutura do sistema tende a degradar conforme novos incrementos são adicionados.
  - ✓ A menos que tempo e dinheiro sejam gastos na reconstrução para melhorar o software, as mudanças regulares tendem a corromper a estrutura do sistema. A incorporação posterior de mudanças no software se torna progressivamente mais difícil e cara.

## Engenharia de software orientada a reúso

- Baseada no reúso sistemático em que os sistemas são integrados com componentes existentes ou sistemas COTS (Commercial-off-the-shelf).
- Estágios do processo:
  - ✓ Análise de componentes;
  - ✓ Modificação de requisitos;
  - ✓ Projeto de sistema com reúso;
  - ✓ Desenvolvimento e integração.
- Atualmente, o reúso é a abordagem padrão para a construção de vários tipos de sistemas de negócio.



## Engenharia de software orientada a reúso

## Tipos de componente de software

- Web services que são desenvolvidos de acordo com padrões de serviço e ficam disponíveis para chamada remota.
- Coleções de objetos que são desenvolvidas como um pacote para ser integrado com um framework como .NET ou J2EE.
- Sistemas de software stand-alone (COTS) que são configurados para uso em ambientes específicos.



## Atividades do processo

- Processos de software reais são sequências intercaladas de atividades técnicas, colaborativas e gerenciais com o objetivo geral de especificar, projetar, implementar e testar um sistema de software.
- As quatro atividades de processo básicas, especificação, desenvolvimento, validação e evolução são organizadas de forma diferente em processos de desenvolvimento distintos.
- No modelo cascata, elas são organizadas em sequências, enquanto no desenvolvimento incremental são intercaladas.

## Especificações de software

- O processo de estabelecer quais serviços são necessários e as restrições na operação e desenvolvimento do sistema.
- Processo de engenharia de requisitos

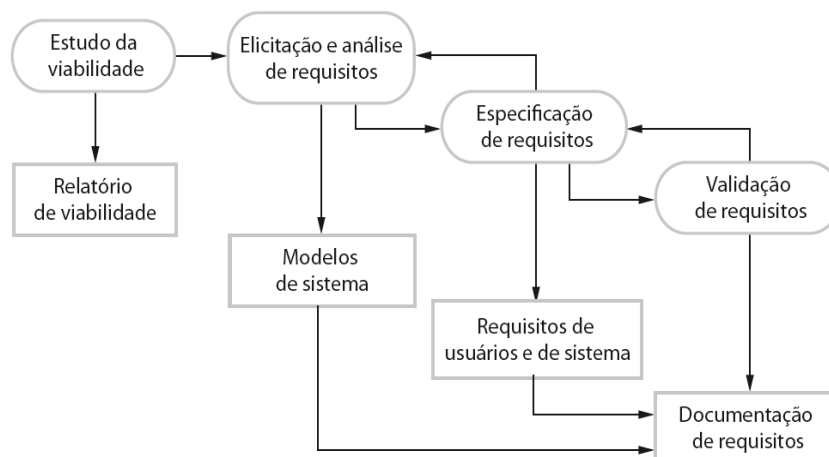
  ✓ Estudo de viabilidade
  É técnica e financeiramente viável construir o sistema?

  ✓ Elicitação e análise de requisitos
  O que os stakeholers do sistema precisam ou esperam do sistema?

  ✓ Especificação de requisitos
  Definição dos requisitos em detalhes.

  ✓ Validação de requisitos
  Verificação da completude dos requisitos.

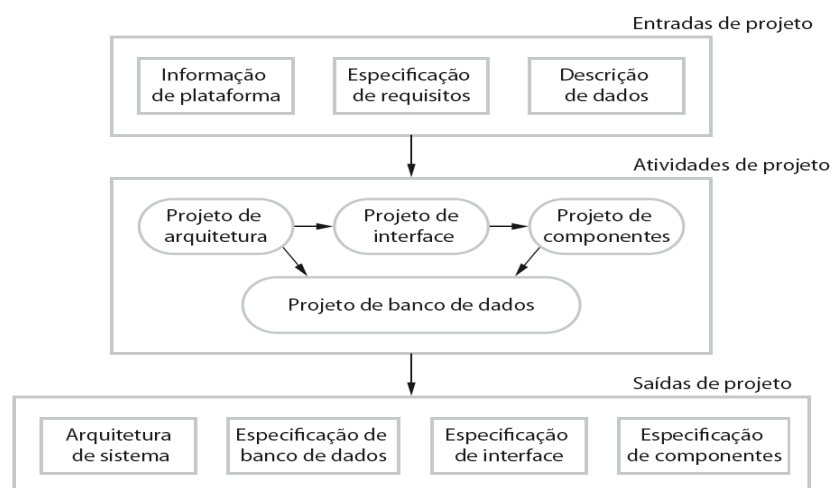
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

## O processo de engenharia de requisitos

Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

## Projeto e implementação de software

- O processo de converter a especificação de sistema em um sistema executável.
- Projeto de software
  - ✓ Design de uma estrutura de software que materialize a especificação;
- Implementação
  - ✓ Transformar essa estrutura em um programa executável;
- As atividades de projeto e implementação são intimamente ligadas e podem ser intercaladas.

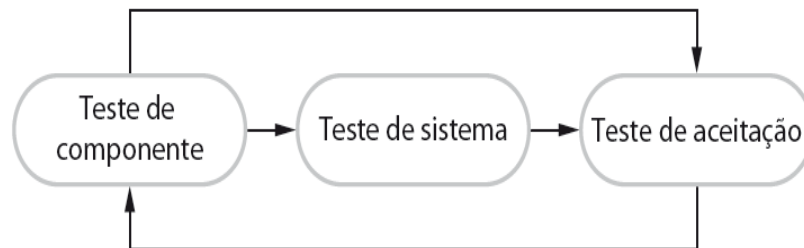


## Modelo geral do processo de design

## Atividades de projeto

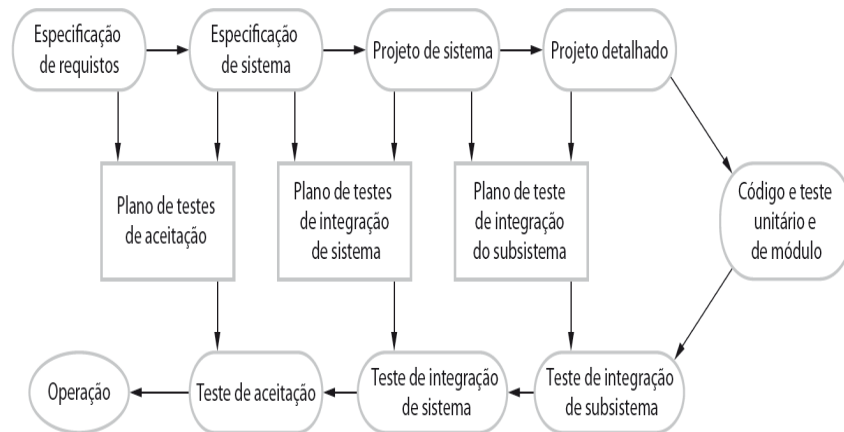
- *Projeto de arquitetura,* em que você identifica a estrutura geral do sistema, os componentes principais (as vezes chamados sub-sistemas ou módulos), seus relacionamentos e como são distribuídos.
- *Projeto de interface,* em que você define as interfaces entre os componentes do sistema.
- *Projeto de componente,* em que você projeta como cada componente do sistema irá operar separadamente.
- *Projeto de banco de dados,* em que você projeta as estruturas de dados do sistema e como essas serão representadas no banco de dados.



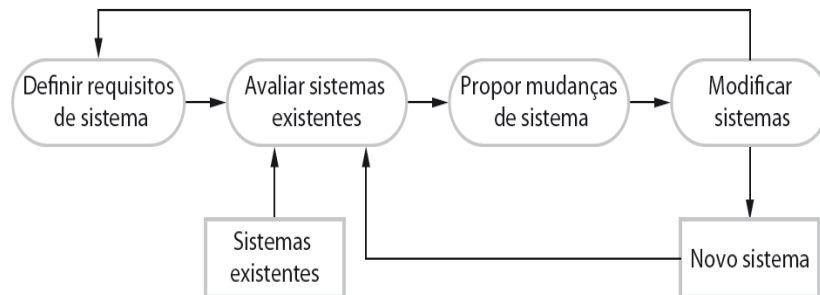
## Validação de software

- Verificação e validação (V & V) serve para mostrar que o sistema está em conformidade com sua especificação e está de acordo com os requisitos do cliente.
- Envolve processos de inspeção e revisão, e testes do sistema.
- Testes do sistema envolvem executar o sistema com casos de teste. São provenientes de especificações dos dados reais que deverão ser processados pelo sistema.
- O teste é a atividade de V & V mais usada.

## Estágios de teste

- Teste de desenvolvimento ou de componente
  - ✓ Componentes individuais são testados independentemente;
  - ✓ Componentes podem ser funções ou objetos , ou agrupamentos coerentes dessas entidades.
- Teste de sistema
  - ✓ Teste do sistema como um todo. Teste de propriedades emergentes são particularmente importantes.
- Teste de aceitação
  - ✓ Teste com dados do cliente para checar se o sistema está de acordo com as necessidades do cliente.

## Fases de teste em um processo de software dirigido a planos

## Evolução do software

- Os softwares são inerentemente flexíveis e podem mudar.
- Conforme os requisitos mudam, conforme mudam as circustâncias do negócio, o software que dá suporte ao negócio também deve evoluir e mudar.
- Apesar de ter acontecido uma demarcação entre desenvolvimento e evolução (manutenção) essa precisa se tornar cada vez mais irrelevante já que tem diminuído a quantidade de sistemas completamente novos.

## Evolução do sistema

## Pontos Importantes

- Os processos de software são as atividades envolvidas na produção de um sistema de software. Os modelos de processo de software são representações abstratas desses processos.
- Modelos de processo gerais descrevem a organização dos processos de software. Exemplos desses processos gerais incluem o modelo 'cascata', desenvolvimento incremental e desenvolvimento orientado a reúso.
- A engenharia de requisitos é o processo de desenvolver uma especificação de software.

## Pontos Importantes

- Processos de projeto e implementação se preocupam em transformar uma especificação de requisitos em um sistema de software executável.
- A validação de software é o processo de checar se o sistema está em conformidade com sua especificação e se esse está de acordo com as necessidades reais do usuário do sistema.
- A evolução de software ocorre quando você altera sistemas de software existentes para adequá-los a novas necessidades. O software precisa evoluir para continuar útil.



## Lidando com mudanças

- As mudanças são inevitáveis em todos grandes projetos de software.
  - ✓ Mudanças no negócio levam a novos e diferentes requisitos de sistema.
  - ✓ Novas tecnologias abrem novas possibilidades para melhorar implementações.
  - ✓ Mudanças de plataforma requerem mudanças na aplicação.
- As mudanças geram retrabalho, o que faz com que o custo das mudanças inclua o retrabalho (p.ex. reanálise dos requisitos) assim como o custo de implementação de novas funções.

## Reduzindo o custo de retrabalho

- Prevenção de mudanças, quando o processo de software inclui atividades que podem antecipar possíveis mudanças antes que o retrabalho se torne necessário.
  - ✓ Por exemplo, um protótipo de sistema pode ser desenvolvido para mostrar algumas características fundamentais do sistema para os clientes.
- Tolerância a mudanças, quando o processo é desenvolvido para que mudanças possam ser acomodadas a um custo relativamente baixo.
  - ✓ Geralmente envolve alguma forma de desenvolvimento incremental. As mudanças propostas podem ser implementadas em incrementos que ainda não foram desenvolvidos. Se isso é impossível, então um incremento único (uma pequena parte do sistema) pode ser alterada para incorporar a mudança.



## Prototipação de software

- Um protótipo é uma versão inicial de um sistema usada para demonstrar conceitos e testar opções de projeto.
- Um protótipo pode ser usado:
  - ✓ No processo de engenharia de requisitos para ajudar na elicitação e validação de requisitos;
  - ✓ Nos processos de projeto para explorar opções e desenvolver um projeto de interface de usuário;
  - ✓ No processo de testes para executar testes fim-a-fim.

## Benefícios da prototipação

- Melhoria do uso do software.
- Maior proximidade com as necessidades do usuário.
- Melhorias na qualidade do projeto.
- Maior manutenibilidade.
- Reduzir esforços de desenvolvimento.



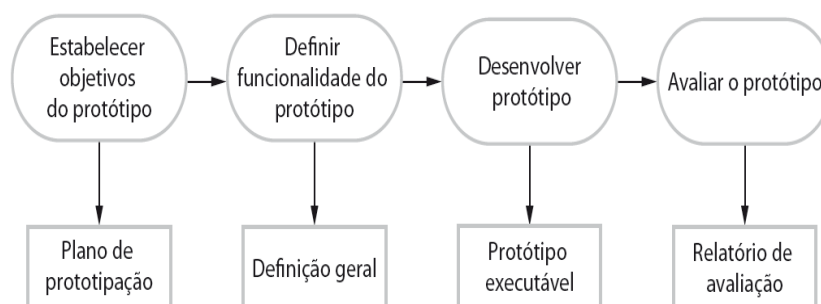
## O processo de desenvolvimento de protótipo

Estabelecer objetivos do protótipo → Definir funcionalidade do protótipo → Desenvolver protótipo → Avaliar o protótipo

Estabelecer objetivos do protótipo ↓ Plano de prototipação

Definir funcionalidade do protótipo ↓ Definição geral

Desenvolver protótipo ↓ Protótipo executável

Avaliar o protótipo ↓ Relatório de avaliação

## Desenvolvimento de protótipos

- Pode ser baseado em linguagens ou ferramentas de prototipagem rápida.
- Pode deixar a funcionalidade de fora do teste.
  - ✓ A prototipação deve focar em áreas do produto que não são bem entendidas;
  - ✓
  - ✓ A checagem de erros e recuperação podem não estar incluídas no protótipo;
  - ✓ O foco deve ser em requisitos funcionais ao invés de não funcionais como por exemplo, a confiabilidade e a segurança.



## Descarte de protótipos

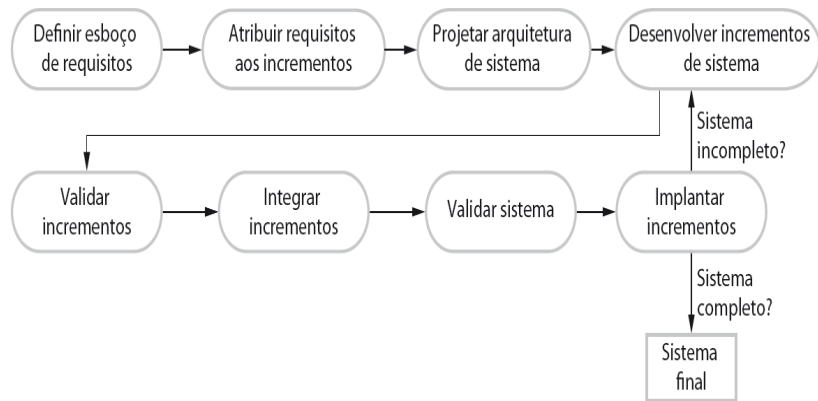
- Os protótipos devem ser descartados depois do desenvolvimento, pois não são uma boa base para um sistema em produção:
  - ✓ Pode ser impossível ajustar o sistema para alcançar requisitos não funcionais;
  - ✓ Geralmente os protótipos não possuem documentação;
  - ✓ Geralmente a estrutura do protótipo é degradada por mudanças rápidas;
  - ✓ Provavelmente o protótipo não irá alcançar os padrões normais de qualidade organizacional.

## Entrega incremental

- Ao invés de entregar o sistema em uma única entrega, o desenvolvimento e a entrega são distribuídos em incrementos, nos quais cada incremento entrega parte da funcionalidade necessária.
- Os requisitos do usuário são priorizados e os requisitos de mais alta prioridade são incluídos nos primeiros incrementos.
- Assim que o desenvolvimento de um incremento é iniciado os requisitos são congelados, mas os requisitos dos incrementos posteriores podem continuar a evoluir.



## Desenvolvimento e entrega incremental

- Desenvolvimento incremental
  - ✓ Desenvolve o sistema em incrementos e avalia cada incremento antes de proceder com o desenvolvimento do próximo incremento;
  - ✓ Abordagem normalmente usada em métodos ágeis;
  - ✓ Avaliação feita por representantes do usuário/cliente.
- Entrega incremental
  - ✓ Implanta um incremento para uso do usuário-final;
  - ✓ Avaliação mais realística sobre o uso prático do software;
  - ✓ Difícil de implementar para sistemas substitutos devido aos incrementos possuírem menos funções do que o sistema que está sendo substituido.

## Entrega incremental

## Vantagens da entrega incremental

- Os valores podem ser entregues ao cliente junto com cada incremento, e funções do sistema ficam disponíveis mais rápidamente.
- Primeiros incrementos agem como protótipos para ajudar a deduzir requisitos para incrementos posteriores.
- Menor risco de falha geral do projeto.
- Os serviços mais prioritários do sistema tendem a serem mais testados.

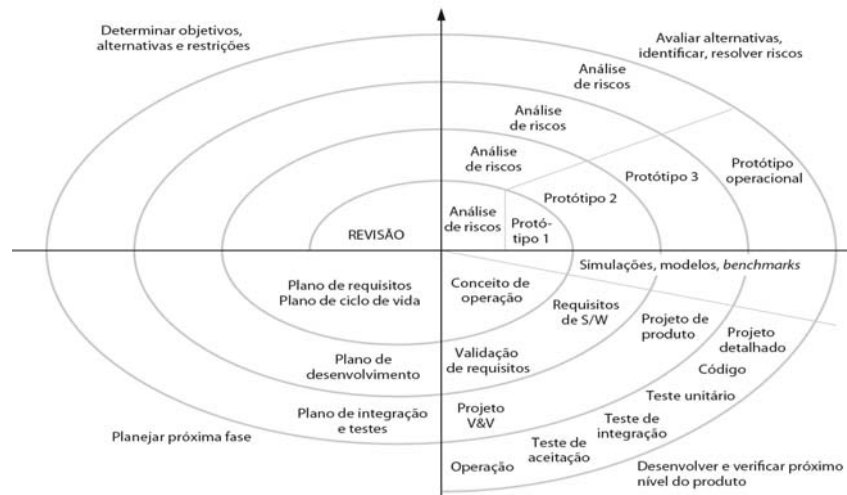
## Problemas da entrega incremental

- A maioria dos sistemas requer um conjunto de funções básicas que são usadas por diferentes partes do sistema.
  - ✓ Como os requisitos não são definidos em detalhes até que um incremento seja implementado, pode ser difícil identificar funções comuns que são necessárias a todos os incrementos.
- A essência dos processos iterativos é que a especificação seja desenvolvida em conjunto com o software.
  - ✓ No entanto, essa pode entrar em conflito com o modelo de aquisição de muitas organizações, nos quais a especificação completa do sistema é parte do contrato de desenvolvimento do sistema.



## Modelo espiral de Boehm

- O processo é representado como uma espiral ao invés de uma sequência de atividades com retornos.
- Cada loop na espiral representa uma fase do processo.
- Não existem fases fixas como especificação ou projeto – os loops na espiral são escolhidos de acordo com a necessidade.
- Os riscos são avaliados explicitamente e resolvidos no decorrer do processo.

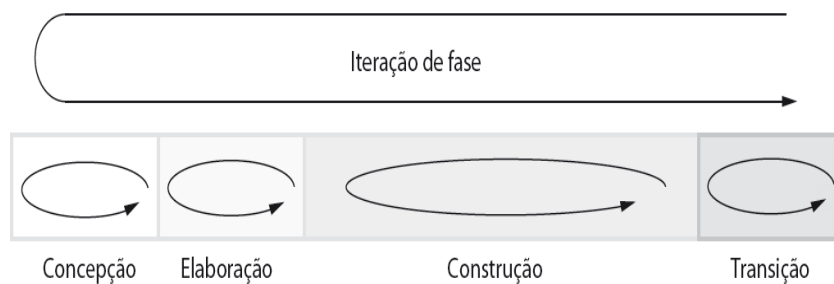
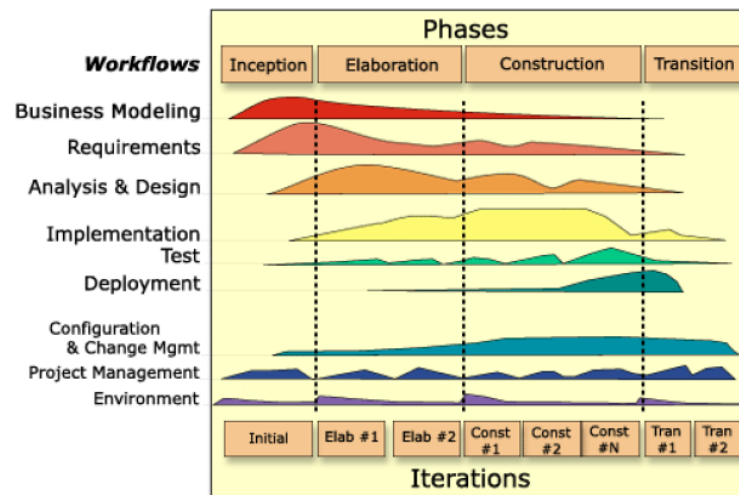
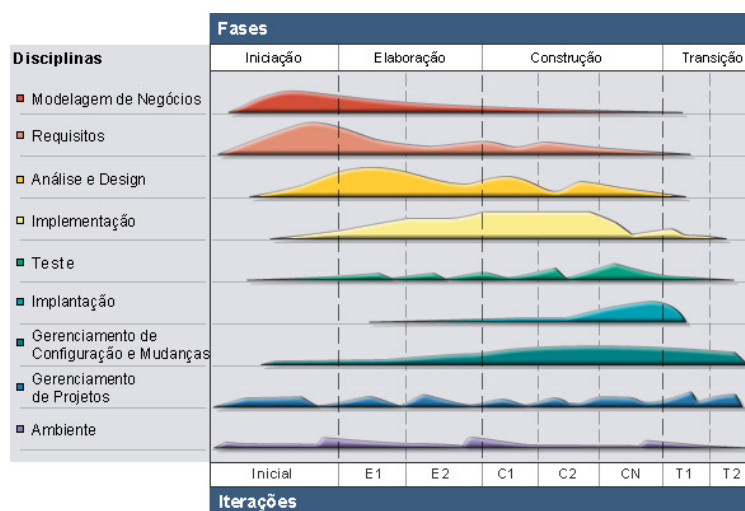
## O modelo de processo de software espiral de Boehm

## Setores do modelo espiral

- Definição de objetivos
  - ✓ São identificados os objetivos específicos para cada fase.
- Avaliação e redução de riscos
  - ✓ Os riscos são avaliados e atividades executadas para reduzir os principais riscos.
- Desenvolvimento e validação
  - ✓ Um modelo de desenvolvimento para o sistema é escolhido, pode ser qualquer um dos modelos genéricos.
- Planejamento
  - ✓ O projeto é revisto e a próxima fase da espiral é planejada.

## Rational Unified Process (RUP)

- É um processo genérico moderno, derivado do trabalho em UML e processos associados.
- Reúne aspectos dos 3 modelos genéricos discutidos previamente.
- Geralmente descrito por 3 perspectivas:
  - ✓ Uma perspectiva dinâmica que mostra fases no tempo;
  - ✓ Uma perspectiva estática que mostra atividades do processo;
  - ✓ Uma perspectiva prática que sugere boas práticas.



## Fases no Rational Unified Process

Rational Unified Process (RUP)
Phases
Workflows
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.
88

Rational Unified Process (RUP)
Fases
Disciplinas
Iniciação
Elaboração
Construção
Transição
Modelagem de Negócios
Requisitos
Análise e Design
Implementação
Teste
Implantação
Gerenciamento de Configuração e Mudanças
Gerenciamento de Projetos
Ambiente
Inicial
E1
E2
C1
C2
CN
T1
T2
Iterações
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

## Fases do RUP

- Concepção (ou iniciação)
  - ✓ Estabelece o *business case* para o sistema.
- Elaboração
  - ✓ Desenvolve um entendimento da extensão do problema e da arquitetura do sistema.
- Construção
  - ✓ Projeta o sistema, programa e testa o sistema.
- Transição
  - ✓ Implanta o sistema no seu ambiente de operação.



## Iteração do RUP

- Iteração Intra-fase
  - ✓ Cada fase é iterativa aos resultados desenvolvidos incrementalmente
- Iteração Inter-fase
  - ✓ Como mostrado pelo loop no modelo RUP, o conjunto todo de fases pode ser executado incrementalmente.

## Workflows estáticos no RUP

| *WORKFLOW* | DESCRIÇÃO |
|---|---|
| Modelagem de negócios | Os processos de negócio são modelados por meio de casos de uso de negócios. |
| Requisitos | Atores que interagem com o sistema são identificados e casos de uso são desenvolvidos para modelar os requisitos do sistema. |
| Análise e projeto | Um modelo de projeto é criado e documentado com modelos de arquitetura, modelos de componentes, modelos de objetos e modelos de sequência. |
| Implementação | Os componentes do sistema são implementados e estruturados em subsistemas de implementação. A geração automática de código a partir de modelos de projeto ajuda a acelerar esse processo. |
| Teste | O teste é um processo iterativo que é feito em conjunto com a implementação. O teste do sistema segue a conclusão da implementação. |



## Workflows estáticos no RUP

| *WORKFLOW* | DESCRIÇÃO |
|---|---|
| Implantação | Um *release* do produto é criado, distribuído aos usuários e instalado em seu local de trabalho. |
| Gerenciamento de configuração e mudanças | Esse *workflow* de apoio gerencia as mudanças do sistema (veja o Capítulo 25). |
| Gerenciamento de projeto | Esse *workflow* de apoio gerencia o desenvolvimento do sistema (veja os capítulos 22 e 23). |
| Meio ambiente | Esse *workflow* está relacionado com a disponibilização de ferramentas apropriadas para a equipe de desenvolvimento de software. |

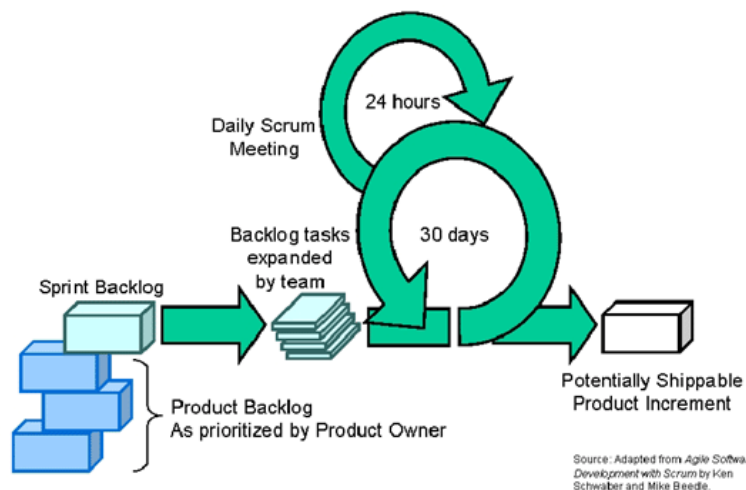
## Boas práticas do RUP

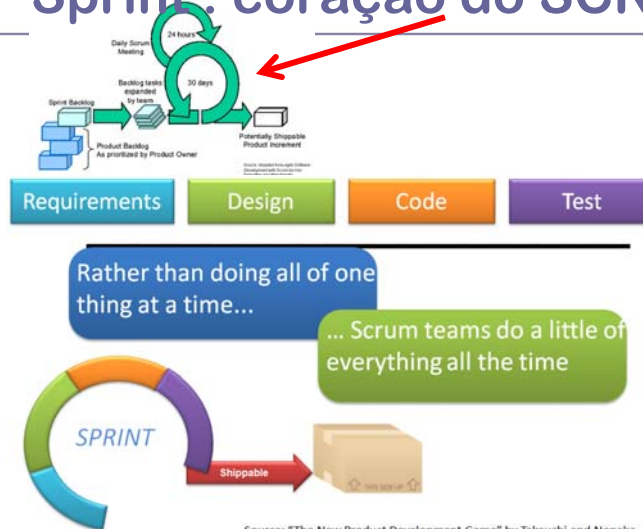
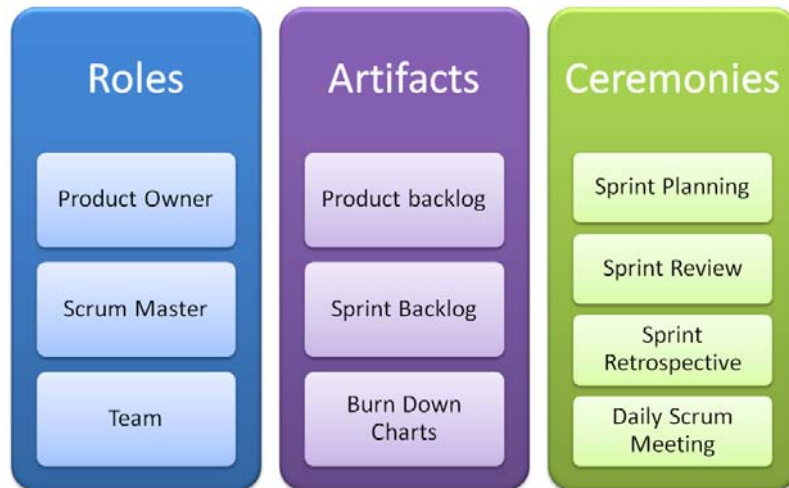
- Desenvolver software iterativamente
  - ✓ Planejar incrementos baseando-se nas prioridades do cliente e entregar as de prioridade mais alta primeiro.
- Gerenciar os requisitos
  - ✓ Documentar explicitamente os requisitos do cliente e manter registros de mudanças desses requisitos.
- Usar arquiteturas baseadas em componentes
  - ✓ Organizar a arquitetura do sistema como um conjunto de componentes reusáveis.



## Boas práticas do RUP

- Modelar o software visualmente
  - ✓ Use modelos de gráficos UML para representar visões dinâmicas e estáticas do software.
- Verificar a qualidade do software
  - ✓ Garanta que o software atenda aos padrões de qualidade organizacional.
- Controlar as mudanças do software
  - ✓ Gerencie as mudanças no software usando um sistema de gerenciamento de mudanças e ferramentas de gerenciamento de configuração.

SCRUM
scrum noun \'skrəm\
Definition of SCRUM
1 a or scrum·mage : a rugby play in which the forwards of each side come together in a tight formation and struggle to gain possession of the ball using their feet when it is tossed in among them; also : the arrangement of players in a scrum
Scrum Seven
Scrum Logo
Scrum
ScrumAlliance
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Scrum Overview
24 hours
Daily Scrum Meeting
Backlog tasks expanded by team
30 days
Sprint Backlog
Potentially Shippable Product Increment
Product Backlog
As prioritized by Product Owner
Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Sprint : coração do SCRUM
Requirements
Design
Code
Test
Rather than doing all of one thing at a time...
... Scrum teams do a little of everything all the time
SPRINT
Shippable
Source: "The New Product Development Game" by Takeuchi and Nonaka. Harvard Business Review, January 1986.
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Mudanças durante o Sprint
SPRINT
Shippable
SCRUM Master
Changes
The Scrum Master ensures that no changes are made that would affect the sprint goal
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Framework SCRUM
Roles
Product Owner
Scrum Master
Team
Artifacts
Product backlog
Sprint Backlog
Burn Down Charts
Ceremonies
Sprint Planning
Sprint Review
Sprint Retrospective
Daily Scrum Meeting
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Papéis - Roles
Product Owner
Define the features of the product
– Product Backlog owner
Decide on release date and content
Be responsible for the profitability of the product (ROI)
Prioritize features according to market value
Adjust features and priority every iteration, as needed
Accept or reject work results
He can cancel an Sprint.
Scrum Master
Represents management to the project
Responsible for enacting Scrum values and practices
Removes impediments
Ensure that the team is fully functional and productive
Enable close cooperation across all roles and functions
Shield the team from external interferences
A Scrum Master should never be the Product owner
Team
Typically 7 people (+/- 2)
Cross-functional team:
Teams are self-organizing
Membership should change only between sprints
Turns the product backlog into increments of potentially shippable functionality
Show the deliverables to the product owner
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

| Backlog item | Estimate |
|---|---|
| Allow a guest to make a reservation | 3 |
| As a guest, I want to cancel a reservation. | 5 |
| As a guest, I want to change the dates of a reservation. | 3 |
| As a hotel employee, I can run RevPAR reports (revenue-per-available-room) | 8 |
| Improve exception handling | 8 |
| ... | 30 |
| ... | 50 |

| Tasks | Mon | Tues | Wed | Thur | Fri |
|---|---|---|---|---|---|
| Code the user interfac | 8 | 4 | 8 | | |
| Code the middle tier | 16 | 12 | 10 | 4 | |
| Test the middle tier | 8 | 16 | 16 | 11 | 8 |
| Write online help | 12 | | | | |
| Write the food class | 8 | 8 | 8 | 8 | 8 |
| Add error logging | | | 8 | 4 | |

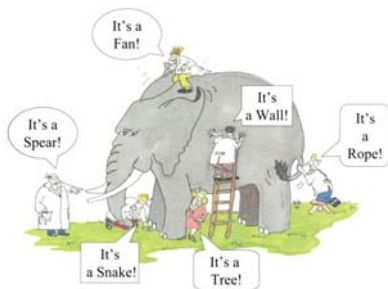
Cerimônias
Sprint Review
This is a 4 hour time-boxed meeting for one month Sprints.
Team presents what it accomplished during the sprint
Typically takes the form of a demo of new features or underlying architecture
The Product Owner identifies what has been done and what hasn't been done.
The Team discusses what went well during the Sprint and what problems it ran into, and how it solved these problems.
Ceremonies
Sprint Planning
Sprint Review
Sprint Retrospective
Daily Scrum Meeting
Sprint Review
Apresentação dos Resultados do Sprint
Demo de novas funcionalidades
Informal ("no slides")
31

Cerimônias
Sprint Retrospective
Whole team gathers and discusses what they'd like to:
Start doing
Stop doing
Continue doing
Ceremonies
Sprint Planning
Sprint Review
Sprint Retrospective
Daily Scrum Meeting
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Cerimônias
Daily Scrum Meeting
Parameters
– Daily
– 15-minutes
– Stand-up
Not for problem solving.
Only team members, Scrum Master, product owner, can talk
Improve communications .
Eliminate other meetings, identify and remove impediments to development
What did you do, since last daily meeting?
What will you do today?
Is there a roadblock on your way?
Ceremonies
Sprint Planning
Sprint Review
Sprint Retrospective
Daily Scrum Meeting

Companies using SCRUM
AGFA HealthCare
Alcatel·Lucent
Allianz
HSBC
Autodesk
EPSON EXCEED YOUR VISION
AUTO SCOUT 24
BT
five
CNN
BOEING
British Gas
Capgemini
YAHOO!
ERICSSON
Google
Microsoft
intel
SIEMENS
NOKIA
myspace a place for friends
SAP
LEGO
hp invent
ING DIRECT
McKinsey&Company
Prof. Flávio de Oliveira Silva, Ph.D.

## Pontos Importantes

- Os processos devem incluir atividades para lidar com mudanças. O que pode envolver uma fase de protipação que ajuda a evitar más escolhas nos requisitos e no projeto.
- Os processos devem ser estruturados para evolução e entrega iterativa, para que as mudanças possam ser feitas sem causar problemas ao sistema como um todo.
- O Rational Unified Process é um modelo de processo genérico moderno, organizado em fases (concepção, elaboração, construção e transição) mas que separa as atividades dessas fases (requisitos, análise e projeto, etc.) .

# Requisitos de Software

## Engenharia de requisitos

- O processo de estabelecer os **serviços** que o cliente necessita do sistema e as **restrições** sob as quais ele opera e é desenvolvido.

- Os próprios requisitos são as descrições dos serviços do sistema e restrições geradas durante o processo de engenharia de requisitos.

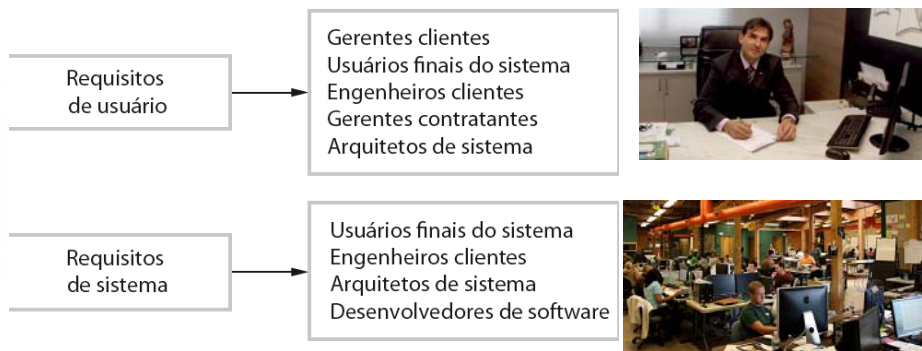
## O que é um requisito?

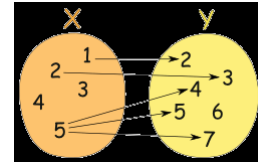
- Pode variar de uma declaração abstrata de alto nível de um serviço ou de uma restrição do sistema para uma especificação matemática funcional.
- Isso é inevitável quando os requisitos podem servir a uma função dupla.
  - ✓ Pode ser a base para a **proposta de um contrato** - portanto, deve ser **aberto** à interpretação;
  - ✓ Pode ser a **base para o contrato em si**, portanto, deve ser definido em **detalhe**;
  - ✓ **Ambas as declarações** podem ser chamadas de requisitos.

## Tipos de requisitos

- **Requisitos de usuário**
  - ✓ Declarações em linguagem natural com diagramas dos serviços que o sistema deverá fornecer e suas restrições operacionais. **Escrito para os clientes**.
- Requisitos de sistema
  - ✓ Um documento estruturado estabelecendo descrições detalhadas das funções do sistema, serviços e restrições operacionais. Define o que deve ser implementado assim, pode ser parte de um **contrato entre o cliente e o empreiteiro**.

Requisitos de usuário e de sistema
Definição de requisitos de usuário
1. O MHC-PMS deve gerar relatórios gerenciais mensais que mostrem o custo dos medicamentos prescritos por cada clínica durante aquele mês.
Especificação de requisitos de sistema
1.1 No último dia útil de cada mês deve ser gerado um resumo dos medicamentos prescritos, seus custos e as prescrições de cada clínica.
1.2 Após 17:30h do último dia útil do mês, o sistema deve gerar automaticamente o relatório para impressão.
1.3 Um relatório será criado para cada clínica, listando os nomes dos medicamentos, o número total de prescrições, o número de doses prescritas e o custo total dos medicamentos prescritos.
1.4 Se os medicamentos estão disponíveis em diferentes unidades de dosagem (por exemplo, 10 mg, 20 mg), devem ser criados relatórios separados para cada unidade.
1.5 O acesso aos relatórios de custos deve ser restrito a usuários autorizados por uma lista de controle de gerenciamento de acesso.
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

Leitores de diferentes tipos de especificação de requisitos
Requisitos de usuário
Gerentes clientes
Usuários finais do sistema
Engenheiros clientes
Gerentes contratantes
Arquitetos de sistema
Requisitos de sistema
Usuários finais do sistema
Engenheiros clientes
Arquitetos de sistema
Desenvolvedores de software
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

# Requisitos funcionais e não-funcionais

- **Requisitos funcionais**
  - O sistema deve fornecer **declarações de serviços**, como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações.
  - Pode explicitar o que o **sistema não deve fazer**.

- **Requisitos não-funcionais**
  - Restrições aos serviços ou funções oferecidas pelo sistema, tais como restrições de tempo, restrições no processo de desenvolvimento, padrões.
  - Muitas vezes se aplica ao sistema como um todo ao invés de características individuais ou serviços.

# Requisitos Funcionais

- Descrever a funcionalidade ou os serviços do sistema.
- Depende do tipo de software, possíveis usuários e o tipo de sistema em que o software é usado.
- Requisitos funcionais **dos usuários** podem ser **declarações de alto nível** a respeito do que o sistema deve fazer.
- Requisitos funcionais **do sistema** devem descrever **detalhadamente** os serviços do sistema.

**Functional Requirements**

1. Image Repository
   - 1.1. Upload a single image into repository
     - 1.1.1. User can select a single image that is stored somewhere on the user's computer (eg. CD, DVD, floppy, hard disk, memory stick) to upload to the repository.
     - 1.1.2. During the selection of an image, the user should have some visual feedback as to what image he is selecting (example, show a thumbnail of the selected image).
     - 1.1.3. After the user selects the image, the user can classify the image by giving values to any number of classifiers
     - 1.1.4. The user can only insert one picture at a time
     - 1.1.5. The user is allowed to insert the same image into the repository many times
     - 1.1.6. Image formats supported by the system are jpg, png and gif.
     - 1.1.7. When the user uploads an image to the repository, a copy of the image is made and saved in the repository.
     - 1.1.8. When the user uploads an image to the repository, a thumbnail format of the copied image is created and saved in the repository.
     - 1.1.9. User can add comments to images
   - 1.2. Search repository for image(s)
     - 1.2.1. Search must be accessible from all levels.
     - 1.2.2. User can search the repository based on classifiers and their desired values (example, **city**: "Vancouver" or **city:** "").
     - 1.2.3. Classifier values for searches can be of any value (use *) or of no value.
     - 1.2.4. User can advance search the repository using logical connectives such as "and" and "or" (example, **city**: "Vancouver" OR **city:** "New Westminster" AND **type of real estate:** "townhouse").
     - 1.2.5. When advance searching, the user can add other classifiers to the search query and the pertinent value that is desired for the search. The search query is connected using logical connectives.
     - 1.2.6. When the user searches the repository, a list of properties that matches the parameters in the query is revealed (user goes to view mode) displaying a thumbnail of the property and classifiers and their values if the value has been set.
   - 1.3. Delete image
     - 1.3.1. User must be able to delete an image from the repository and should be prompted as to what listings the image is related to.
2. Classifiers
   - 2.1. The name of the default classifiers already defined by the system are
     - city
     - type of real estate [values can be house, condo or townhouse]
     - price range
     - date to market (date at which real estate was put on the market)

# Imprecisão de requisitos

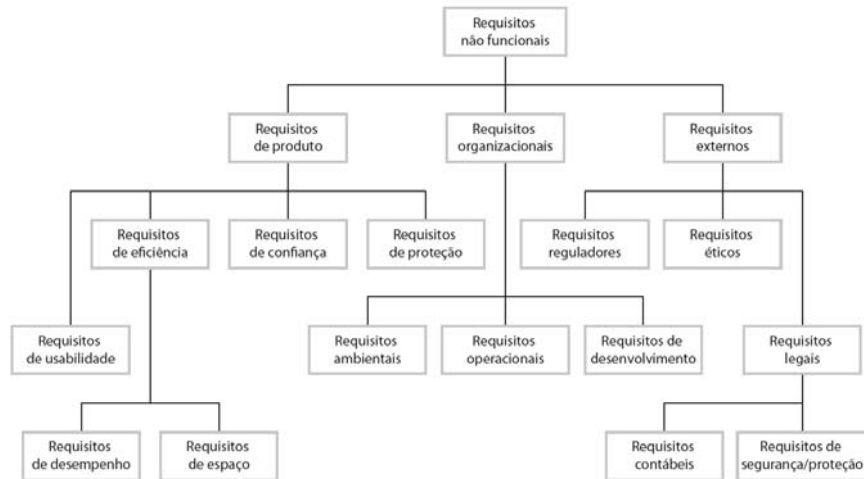
- **Problemas** surgem quando os requisitos não são precisamente definidos.
- **Requisitos ambíguos** podem ser interpretados de maneiras diferentes por desenvolvedores e usuários.
- Considere o termo '**pesquisa**' no requisito 1

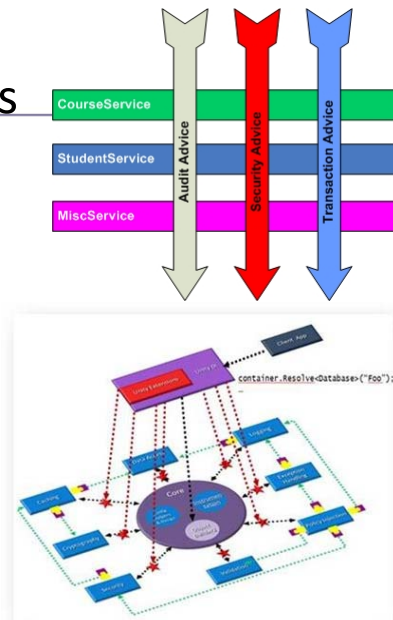
> • Um usuário deve ser capaz de pesquisar as listas de agendamentos para todas as clínicas.

  - ✓ A intenção do usuário – busca pelo nome de um paciente em todos as consultas em todas as clínicas;
- Interpretação do desenvolvedor – busca pelo nome de um paciente em uma clínica. O usuário escolhe a clínica e em seguida pesquisa.



# Requisitos Não-funcionais

- Esses requisitos definem as **propriedades e as restrições** do sistema por exemplo, confiabilidade, tempo de resposta e ocupação de área.
- As restrições são capacidades de dispositivos de E/S, as representações do sistema, etc.
- Os requisitos de processo também podem ser especificados impondo um **IDE particular**, linguagem de programação ou método de desenvolvimento.
- Os requisitos não-funcionais **podem ser mais críticos** do que os requisitos funcionais. Se esses não forem atendidos, o sistema pode ser inútil.

Tipos de requisitos não funcionais
Requisitos não funcionais
Requisitos de produto
Requisitos organizacionais
Requisitos externos
Requisitos de eficiência
Requisitos de confiança
Requisitos de proteção
Requisitos reguladores
Requisitos éticos
Requisitos de usabilidade
Requisitos ambientais
Requisitos operacionais
Requisitos de desenvolvimento
Requisitos legais
Requisitos de desempenho
Requisitos de espaço
Requisitos contábeis
Requisitos de segurança/proteção
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.


Implementação de requisitos não funcionais
Requisitos não-funcionais podem afetar a arquitetura geral de um sistema, em vez de componentes individuais.
Por exemplo, para assegurar que os requisitos de desempenho sejam cumpridos, você pode ter que organizar o sistema para minimizar a comunicação entre os componentes.
Um único requisito não-funcional, como um requisito de proteção, pode gerar uma série de requisitos funcionais relacionados que definem os serviços do sistema que são necessários.
Ele também pode gerar requisitos que restringem os requisitos existentes.
CourseService
StudentService
MiscService
Audit Advice
Security Advice
Transaction Advice
container.Resolve<Database>("Foo");
Modelagem de Software
Prof. Flávio de Oliveira Silva, Ph.D.

# Classificações de requisitos não funcionais

- Requisitos de produto
  - ✓ Requisitos que especificam que o produto entregue deve se comportar de uma maneira particular, por exemplo velocidade de execução, confiabilidade, etc.

- Requisitos organizacionais
  - ✓ Requisitos que são consequência de políticas e procedimentos organizacionais, por exemplo padrões de processo usados, requisitos de implementação, etc.

- Requisitos externos
  - ✓ Requisitos que surgem de fatores externos ao sistema e seu processo de desenvolvimento, por exemplo, requisitos de reguladores, requisitos legais, etc.

# Metas e requisitos

- Requisitos não-funcionais podem ser muito **difíceis de se definir precisamente** e requisitos imprecisos podem ser **difíceis de se verificar**.
- Metas
  - ✓ **A intenção geral do usuário**, facilmente usável.
- Requisito não-funcional **mensurável**.
  - ✓ Uma declaração usando **alguma métrica** que pode ser **objetivamente testada**.
- Metas são **úteis para desenvolvedores** quando exprimem as **intenções dos usuários do sistema**.

# Métricas para especificar requisitos não funcionais

| Propriedade | Medida |
|---|---|
| Velocidade | Transações processadas/segundo<br>Tempo de resposta de usuário/evento<br>Tempo de atualização de tela |
| Tamanho | Megabytes<br>Número de chips de memória ROM |
| Facilidade de uso | Tempo de treinamento<br>Número de *frames* de ajuda |
| Confiabilidade | Tempo médio para falha<br>Probabilidade de indisponibilidade<br>Taxa de ocorrência de falhas<br>Disponibilidade |
| Robustez | Tempo de reinício após falha<br>Percentual de eventos que causam falhas<br>Probabilidade de corrupção de dados em caso de falha |
| Portabilidade | Percentual de declarações dependentes do sistema-alvo<br>Número de sistemas-alvo |

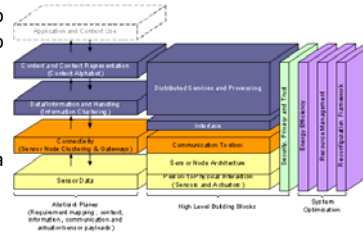


# Especificação de requisitos

- O processo de escrever os requisitos de usuário e de sistema em um documento de requisitos.
- Os requisitos precisam ser **compreensíveis para usuários finais e clientes** que não têm formação técnica.
- Requisitos de sistema são mais detalhados e podem incluir informações mais técnicas.
- Os requisitos podem ser parte de um contrato para o desenvolvimento do sistema.
  - ✓ Portanto, é importante que esses sejam tão completos quanto possível.

# Projeto e requisitos

- Em princípio, os requisitos devem indicar o que o sistema deve fazer e o projeto deve descrever como fazer isso.

- Na prática**, os requisitos e o projeto são inseparáveis**
  - ✓ A arquitetura do sistema pode ser projetada para estruturar os requisitos;
  - ✓ O sistema pode interoperar com **outros sistemas** que restringem o projeto e **impõem requisitos** sobre o novo sistema;
  - ✓ O uso de uma arquitetura específica para satisfazer os requisitos não funcionais pode ser um requisito de domínio.



# Diretrizes para escrever requisitos

- DEFINIR um **formato padrão** e usá-lo para **todos** os requisitos.
- Usar a linguagem de uma forma consistente.
- Usar ‘**deve**’ para requisitos obrigatórios e ‘**pode**’ para os requisitos desejáveis.
- Usar o **realce de texto** para identificar as **partes fundamentais** do requisito.
- **Evitar o uso de jargões** de computador.
- Incluir uma **justificativa (lógica)** de por que um requisito é necessário.

O processo de elicitação e análise de requisitos
1. Descoberta de requisitos
2. Classificação e organização de requisitos
3. Priorização e negociação de requisitos
4. Especificação de requisitos
elagem de Software
avio de Oliveira Silva, Ph.D.